Politechnika Krakowska
Wydział Fizyki, Matematyki i Informatyki
Programowanie Rozproszone i Równoległe

Wyszukiwanie wzorca w tekście

Autory:
Łukasz Gonciarz
Matusz Kalinowski

19 stycznia 2015

Cel pracy

Celem wykonanej pracy jest zapoznanie się z programowaniem w środowisku z wymiana komunikatów miedzy procesami MPI.

Message Passing Interface (MPI)

Message Passing Interface (MPI) protokół komunikacyjny będący standardem przesyłania komunikatów pomiędzy procesami programów równoległych działających na jednym lub więcej komputerach. Celami MPI są wysoka jakość, skalowalność oraz przenośność. MPI jest dominującym modelem wykorzystywanym obecnie w klastrach komputerów oraz superkomputerach. Pierwsza wersja standardu ukazała się w maju 1994 r.

Wyszukiwanie wzorca w tekscie

Problem z jakim postanowiliśmy się zmierzyć jest wyszukiwanie wzorca w tekście. Problem ten jest rozwiązywany przez każdego człowieka codziennie wiele razy. Do realizacji wybraliśmy algorytm Karpa Rabina. Uchodzi on za jeden z najwydajniejszych algorytmów wśród stosowanych do wyszukiwania wzorca w tekście.

Algorytm KarpaRabina został stworzony w 1987 roku przez Michaela O. Rabina i Richarda Karpa. Dziś algorytm ten najczęściej znajduję zastosowanie w problemie jakim jest wyszukiwanie plagiatów. Algorytm ten w pseudo kodzie został zaprezentowany na rysunku nr 1.

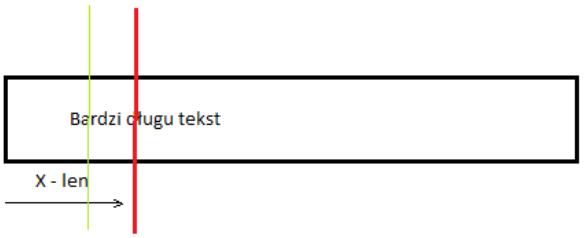
Podział pliku dla poszczególnych procesów

```
1 function RabinKarp(string s[1..n], string sub[1..m])
2     hsub := hash(sub[1..m])
3     hs := hash(s[1..m])
4     for i from 1 to n
5         if hs = hsub
6             if s[i..i+m-1] = sub
7                 return i
8         hs := hash(s[i+1..i+m])
9     return not found
```

Rysunek 1: Algorytm Karpa Rabina

Kolejnym wyzwaniem z jakim należało się zmierzyć jest odpowiedni podział pliku oraz przesłanie odpowiedniej porcji analizowanych danych (pliku) poszczególnym procesom. Stanowi to nie lada problem. Ponieważ należy pamiętać o pewnym offsecie danych tak by proces poszukiwania rozwiązania miał sens. Na rysunku nr 2 ilustruje sposób myślenia jaki analizowaliśmy w celu osiągnięcia przez nas zamierzonego celu

Jeśli mamy bardzo długi tekst i ustalimy, że rozmiar przesyłanych danych wynosi Z oraz długość wzorca to X, należy zapamiętać index na którym skończyliśmy oraz długość cofnąć się długość wzorca X - 1. Ten sposób myślenia doprowadził nas do sukcesu jakim jest działający poprawnie i bardzo szybko program.



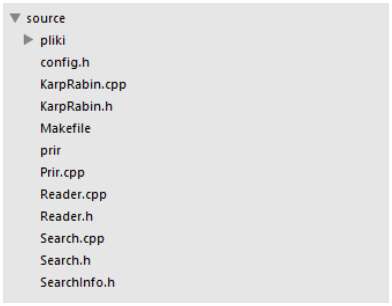
Rysunek 2: Algorytm Karpa Rabina

Implementacja

Program napisano przy wykorzystaniu środowiska MPI, w języku programowania c++, zdecydowaliśmy się na ten język programowania ze względu na fakt, iż jest on nieporównywalnie wydajniejszy w porównaniu do Javy, która rozważaliśmy na początku.

Na rysunku nr 3 przedstawiona jest struktura projektu.

Najważniejsze pliki to Prir.cpp oraz Search.cpp gdzie zaimplementowano całą logikę związaną z procesem zrównoważenia zadania. Na uwagę zasługuję metoda run gdzie odbywa się proces wysyłania rysunek nr 4 przedstawia ową metodę. Istotna także jest metoda Wait, która to jest z kolei odpowiada za odnalezienie wzorca w badanych źródle (rysunek nr. 5).



Rysunek 3: Struktura projektu

```
32  void Search::Run(){
33
34      ResetProcessesInfo();
35
36      for (map<string, string>::iterator it = files.begin(); it != files.end(); ++it)
37      {
38          //otwieramy plik
39          ifstream file(it->second.c_str());
40
41          char pack[packSize+1];
42          char packTemp[packSize-overlapSize+1];
43
44          //wczytujemy pierwszą paczkę
45          file.get(pack, packSize+1);
46          //kolejne paczki
47
48          SendToProcess(pattern, pack, it->second);
49
50          while(file.get(packTemp, packSize-overlapSize+1)){
51              for(unsigned int i=0; i<overlapSize; i++){
52                  pack[i] = pack[packSize-overlapSize+i];
53              }
54
55              for(unsigned int i=overlapSize; i<packSize; i++){
56                  pack[i] = packTemp[i-overlapSize];
57              }
58              SendToProcess(pattern, pack, it->second);
59          }
60      }
61
62      SendBreakInfo();
63  }
```

Rysunek 4: Metoda Run

Należy również podkreślić, iż proces główny odpowiada w naszym przypadku za przesył danych, na pozostałych procesach z wykluczeniem procesu głównego, gdzie odbywają się obliczenia.

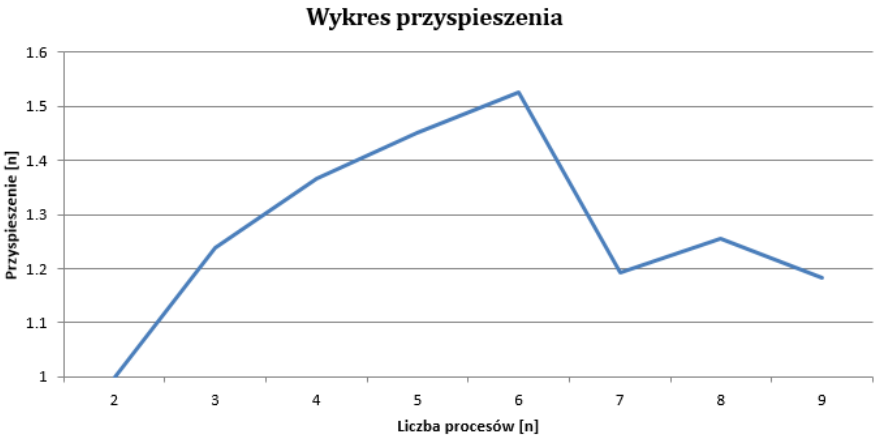
```
128 //Metoda dla procesów innych niż główny - czeka na dane i wykonuje na nich wyszukiwanie
129 void Search::Wait(){
130     SearchInfo sInfo;
131
132     int processNumber;
133     MPI_Comm_rank(MPI_COMM_WORLD, &processNumber);
134     while(true){
135         MPI_Recv(&sInfo, 1, *MPI_SearchInfo, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, MPI_STATUSES_IGNORE);
136
137         if(sInfo.pattern[0] == 0){
138             //cout << "Proces " << processNumber << "konczy działanie" << endl;
139             break;
140         }
141
142         KarpRabin karp;
143         int q = 101;
144         int alfabeath = 256;
145         int value = karp.karp_rabin(sInfo.pattern, sInfo.text, q, alfabeath);
146
147         if(value>0){
148             cout << "Fraza znaleziona w pliku: " << sInfo.fileName << endl;
149         }
150         MPI_Send(&processNumber, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
151     }
152 }
```

Rysunek 5: Metoda Wait

Pomiary

Poniższa sekcja prezentuję uzyskane wyniki Rysunek nr 6 prezentuję przyspieszenie jakie udało się uzyskać dla wyszukiwania w plikach o łącznym rozmiarze 90 MB, składających się z 8 plików, jest to bardzo ważne ponieważ podczas przetwarzania zostało otwartych 8 plików, a jak wiadomo taka operacja potra-

fi być kosztowna. Rysunek nr 7 przedstawia
zależność czasu od rozważanej ilości procesów. Jak widać, na wykresach obliczenia uzyskują przyspieszenie do 6 procesów, dalsze zwiększanie ilości procesów powoduję duży narzut na komunikację prze co przyspieszenie spada.



Rysunek 6: Przyspieszenie

Rysunek 7: Zależność czasu od ilości procesów

Przeprowadzone zostały dalsze badania, które są zaprezentowane poniżej. Tym razem postanowiliśmy pokazać wyniki dla pojedynczego pliku o rozmiarze 90 MB, przetważanie takiego pliku zajowało średnio 500 ms zatem widzimy w tym wypadku jak kosztowna jest operacja I/O.

Wnioski

- Algorytm KarpaRabina pozwala na bardzo wydajne wyszukiwanie ciągów w tekście
- Komunikacja między procesowa jest bardzo kosztowna, jest wąskim gardłem w obliczeniach równoległych
- Podczas badań jakie przeprowadziliśmy zaobserwowaliśmy brak zwrostu wydajności powyżej 6 procesów dla badanego przypadku, wpływ ma na to przede wszystkim fakt, że proces główny nie nadąża z wysyłaniem paczek, prze co procesy pozostają bezrobotne i oczekują na dane
- Wpływ na osiągnięte rezultaty ma również zależności między plikami jakie występują podczas podziału danych
- Udało nam się udowodnić sensowność programowania równoległego, zmniejszyliśmy czas o 1/3 co w przypadku takich operacji jakie są wykonywane na co dzień może mieć duże znaczenie
- Dodatkową poprawę wydajności można uzyskać ograniczając ilość przesyłanych danych między procesami.